1999-09

Operational maneuver from the sea logistics training aid

Sterba, John R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

OPERATIONAL MANEUVER FROM THE SEA LOGISTICS TRAINING AID

by

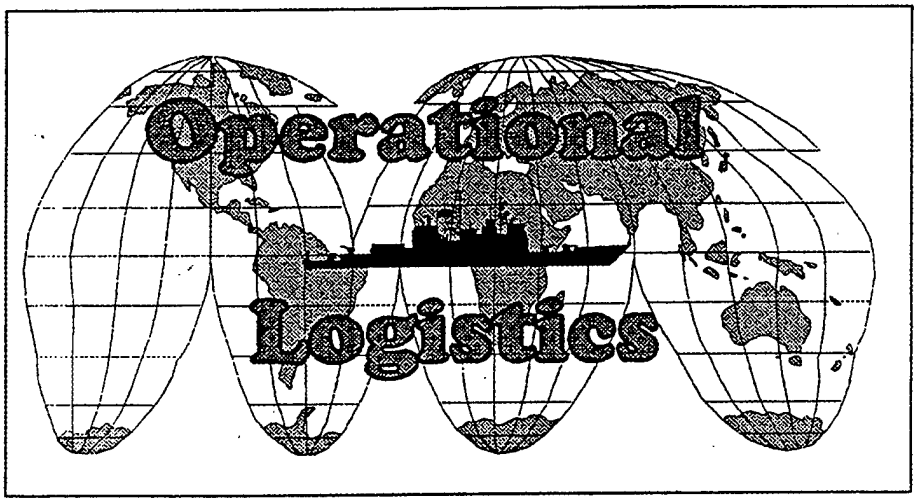
John R. Sterba

September 1999

Thesis Advisor: Arnold Buss
Second Reader: Kevin J. Maher

Aproved for public release; distribution is unlimited.

Amateurs discuss strategy, Professionals study logistics



REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1999	Master's Thesis

4. TITLE AND SUBTITLE
OPERATIONAL MANEUVER FROM THE SEA LOGISTICS TRAINING AID

5. FUNDING NUMBERS

6. AUTHOR(S)
Sterba, John R.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Changes in the global situation resulting from the demise of the Soviet Union have led to major restructuring in United States' military. As a result the Navy and Marine Corps have reevaluated amphibious concepts, resulting in Operational Maneuver From the Sea (OMFTS), and Sea Based Logistics (SBL). OMFTS supported by SBL rely on a "sea-base" to act as the staging area for the dissemination of supplies to the amphibious units ashore. This close integration of Navy and Marine Corps logistics requires Naval Officers and Marine Corps Officers be aware of each other's requirements.

Current training aids do not model OMFTS and SBL. To facilitate training a modular computer-based wargame called NAVLOGS has been developed with the logistics concerns of OMFTS and SBL in mind. The core classes of NAVLOGS include the Scenario Builder, the Coastline Model, and a Map Scale Generator. These core classes enable the user to create scenarios for NAVLOGS with little or no programming skill thus taking scenario design from the programmers and giving it to the end user.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Operational Maneuver From the Sea, Naval Expeditionary Logistics, NAVLOGS, Sea-Based Logistics, Ship to Objective Maneuver	72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

OPERATIONAL MANEUVER FROM THE SEA LOGISTICS TRAINING AID

John R. Sterba
Lieutenant, United States Navy
B. S., Ohio State University, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1999

Author: [redacted]
John R. Sterba

Approved by: [redacted]
Arnold Buss, Thesis Advisor

[redacted]
Kevin J. Maher, Second Reader

[redacted]
Richard E. Rosenthal, Chairman
Department of Operations Research

ABSTRACT

Changes in the global situation resulting from the demise of the Soviet Union have led to major restructuring in United States' military. As a result the Navy and Marine Corps have reevaluated amphibious concepts, resulting in Operational Maneuver From the Sea (OMFTS), and Sea Based Logistics (SBL). OMFTS supported by SBL rely on a "sea-base" to act as the staging area for the dissemination of supplies to the amphibious units ashore. This close integration of Navy and Marine Corps logistics requires Naval Officers and Marine Corps Officers be aware of each other's requirements.

Current training aids do not model OMFTS and SBL. To facilitate training a modular computer-based wargame called NAVLOGS has been developed with the logistics concerns of OMFTS and SBL in mind. The core classes of NAVLOGS include the Scenario Builder, the Coastline Model, and a Map Scale Generator. These core classes enable the user to create scenarios for NAVLOGS with little or no programming skill thus taking scenario design from the programmers and giving it to the end user.

THESIS DISCLAMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAAV	Advanced Assault Amphibian Vehicle
ACE	Air Combat Element
ARG	Amphibious Readiness Group
CAS	Close Air Support
CE	Command Element
CSSE	Combat Service Support Element
GCE	Ground Combat Element
GUI	Graphic User Interface
HMMWV	High Mobility Multipurpose Wheeled Vehicle
LAV	Light Armored Vehicle
LFORM	Landing Force Munitions
LHD	Landing Helicopter Dock
LPD	Landing Platform Dock
LSD	Landing Ship Dock
MEU	Marine Expeditionary Unit
NEL	Naval Expeditionary Logistics
NPS	Naval Postgraduate School
OMFTS	Operational Maneuver From the Sea
OOP	Object-Oriented Programming
SAM	Surface to Air Missile
SBL	Sea Based Logistics
STOM	Ship To Objective Maneuver

EXECUTIVE SUMMARY

The end of the Cold War in the 1990s forced a restructuring of the United States military, resulting in the naval leadership re-evaluating its role as a combat force. With the traditional blue water threat of the Soviet Navy gone, the vision of the 21st century fleet was redefined. Littoral warfare is the Navy's area of interest in the future. The vast majority of the world's population centers and most of the natural resources are located in the coastal regions. With the increased effects of the global economy felt throughout the world, the United States will inevitably face conflicts within these regions. To combat this threat the Marine Corps and the Navy developed Operational Maneuver From the Sea (OMFTS), and Naval Expeditionary Logistics (NEL).

The Marine Corps' OMFTS relies on a naval sea-base and the Sea-Based Logistics of NEL. By using the sea-base the Marines can strike out onto the land in a nonlinear fashion without having to defend a logistics depot. In nonlinear warfare Marines use aerial insertion and amphibious craft to strike the objective directly from the transport ships, a form of maneuver known as Ship To Object Maneuver (STOM). STOM is highly flexible and responsive to rapid changes in the combat environment. It also frees units from the dangerous task of defending the over land routes to a logistics depot. The Marine Corps is incorporating these new concepts into their doctrine and will rely on the Navy to provide adequate logistics support. The ships providing the sea-base for the Marine unit ashore must be aware of its requirements and their relative priority. Naval and Marines Corps logisticians need to be exposed to each other's requirements if they are to function in concert.

Most Naval officers are not familiar with the needs of a Marine unit ashore. However, for OMFTS to function properly Naval officers must have a clear understanding of the staging and organization of Marine ammunition and fuel. Shipboard officers must also be able to take into account the need for food, water, and other supplies that Naval officers do not normally consider. On the other hand Marine Corps logistics officers should become familiar with the difficulties in acquiring supplies while at sea. If officers from both services are aware of the difficulties faced by each other, then OMFTS has an increased probability of succeeding.

Logistics officers can only gain the necessary familiarity with each other's difficulties by working with each other. This can be accomplished by training exercises known as wargames. Wargaming teaches officers the lessons of war during peacetime. Computer based wargames do so without the added expense of staging troops and supplies. A logistics wargame that takes into account OMFTS and Sea-Based Logistics is needed to teach Naval and Marine Corps officers the difficulties that will be faced.

This thesis lays the groundwork for an adaptable computer-based wargame called NAVLOGS that is designed to implement OMFTS and SBL. A fully functioning NAVLOGS application is beyond the scope of this thesis alone. NAVLOGS' ultimate objective is to function as a teaching aid and introduce the user to the difficulties in maintaining the supply lines for an amphibious operation.

This thesis develops a Scenario Builder and a Graphic User Interface (GUI) that allows easy intuitive construction of scenarios for NAVLOGS. To augment the pliability of the Scenario Builder, coastline modeling and model scaling applications have been created. Finally

a database containing the Marine Corps units and their characteristics has been developed that enables flexible model configuration.

In summary the development of the core classes of NAVLOGS allow for an adaptable logistics training aid. Its design allows for upgrades in weapons systems as well as changes in operational concepts. The modular construction of the code itself permits upgrading of NAVLOGS' classes. NAVLOGS will facilitate the training of future logistics officer in the common pitfalls associated with OMFTS and SBL.

I. INTRODUCTION

A. BACKGROUND

The end of the Cold War in the 1990s forced global restructuring of the United States military. As a result naval leadership re-evaluated its role as a combat force. The traditional blue water threat of the Soviet Navy eroded rapidly, and the vision of the 21st century fleet was redefined. The white papers titled ...*From the Sea* (Dept. of the Navy 1992) and *Forward...From the Sea* (Dept. of the Navy 1994) were adopted as the framework for the next century's fleet. *From the Sea* and *Forward... From the Sea* stressed littoral warfare as the Navy's area of interest in the future.

Since the restructuring of the armed forces in the 1990s the United States completed a massive draw down in its military size. While the lack of a Soviet threat diminished the previous polarization of the world between West and East, it has also increased the difficulty of predicting future threats to the US over the Cold War era. There will be little or no warning of future conflicts, so it is impossible to predict exactly where to preposition supplies. Thus, without a known threat, there is a much greater need for transportation assets for logistics than in the past.

The vast majority of the world's population centers and most of the natural resources are located in the coastal regions. With the increased effects of the global economy felt throughout the world, the United States will inevitably face conflicts within these regions. Because there is no longer an open ocean threat, the United States Navy now faces an increase in its participation in littoral conflicts. Additionally the recent proliferation of nuclear weapons in Southern Asia and the sale of former Soviet arms make combat in the littorals more hazardous than in the past.

Therefore, as discussed in *Naval Expeditionary Logistics*, in order to maintain global stability the United States Navy is shifting tactical emphasis by moving away from a blue-water fleet to power projection ashore and the rapid deployment and logistical support of landing forces. (Naval Expeditionary Logistics Committee 1999) The landing forces will rely entirely on Naval Logistics for their initial support.

Naval Logistics is more than a force multiplier; it is the umbilical chord that sustains the Navy and Marine Corps on their overseas deployments. Simply stated, logistics is the underlying infrastructure that provides fuel, food and other critical resources for operations undertaken. In today's joint environment the Navy may be called upon to support Army or Air Force units as well as Naval and Marine Corps. Military operations cannot succeed without logistics; they are as vital to the mission as people and weapons. The Navy and Marine Corps require readily available logistics to fulfill their missions and simply cannot function without the right supplies at the right time.

There are four dimensions to effective logistics operations: availability, maintainability, mobility, and sustainability. Availability is defined as providing the right equipment at the right time and the right place to support the operations effectively. Maintainability is the ability to provide spare parts, support and technical documentation to enable the equipment to function at its highest level throughout its life cycle. Mobility is the ability to move equipment, people and supplies to their desired destination as quickly and efficiently as possible. Finally, sustainability is the requirement to maintain high levels of support allowing the operation to continue unimpeded. The relative importance of these four dimensions of logistics is different today than in the past. (Naval Expeditionary Logistics Committee 1999)

Logistical support for Amphibious warfare dates back at least to Alexander the Great who used ships as floating logistics bases for his troops in the Mediterranean. The majority of the world turned its back on amphibious assault after tragedies like Galipoli in World War I. However, in World War II the United States proved that amphibious warfare could succeed. In its island hopping campaign in the Pacific, the United States relied on a massive build up of supplies ashore for support. Recently however there has been a major paradigm shift in Naval and Marine Corps planning.

Due to the United States increased involvement in low-intensity regional conflicts quick action forces have been required. This new concept called Sea-Based Logistics (SBL) foregoes the build up of a large logistics base on land in favor of a more flexible strike force that is resupplied directly from the "sea-base." Interestingly, SBL is remarkably similar to Alexander's use of the seas. SBL is integral to the Marine Corps' Operational Maneuver From the Sea (OMFTS).

B. OMFTS

Marines strike out onto the land in a nonlinear fashion from a sea-base in OMFTS. The sea-base frees the Marines from having to defend a logistics depot on land. In nonlinear warfare Marines use aerial insertion and amphibious craft to strike the objective directly from the transport ships, a form of maneuver known as Ship To Object Maneuver (STOM). STOM is highly flexible and responsive to rapid changes in the combat environment. It also frees units from the dangerous task of defending the overland routes to a logistics depot. The Marine Corps is incorporating these new concepts into their doctrine and will rely on the Navy to provide adequate logistics support.

In order for OMFTS and STOM to succeed, logistics for expeditionary warfare must undergo restructuring. Marine units are now expected to penetrate up to 200 miles inland. In order for the Marines to achieve extended mobility without a large land-based depot OMFTS must be properly implemented, and the Navy must support it with suitable ships and aircraft. The staging of supplies and ammunition must now take place onboard the ships that are supporting the units ashore. The ships providing the sea base for the Marine unit ashore must be aware of its requirements and their relative priority. Naval and Marines Corps logisticians need to be exposed to each other's requirements if they are to function in concert.

Future Marine Corps campaign successes will critically depend on precise calculations and flawless timing from all elements of Naval Logistics. This dependence will require Naval Logistics officers to assume vital new roles throughout the chain of command. Most Naval officers are not familiar with the needs of a Marine unit ashore. However, for OMFTS to function properly Naval officers must have a clear understanding of the staging and organization of Marine ammunition and fuel. Shipboard officers must also be able to take into account the need for food, water, and other supplies that Naval officers do not normally consider. On the other hand, Marine Corps logistics officers should become familiar with the difficulties in acquiring supplies while at sea. By training officers from both services to the difficulties faced by each other OMFTS has an increased probability of succeeding. The underlying goal of this thesis is to develop a training aid that is used by both the Navy and Marine Corps.

Logistics officers can only gain the necessary familiarity with each other's difficulties by working with each other. This can be accomplished by training exercises known as wargames. Wargaming teaches officers the lessons of war during peacetime. Computer based wargames do

so without the added expensive of staging troops and supplies. A logistics wargame that takes into account OMFTS and Sea-Based Logistics is needed to teach Naval and Marine Corps officers the difficulties that will be faced. PROLOG, a logistics wargame that models the support of a carrier battle group, is currently in use at the Naval Postgraduate School (NPS).

C. PROLOG

In May 1982, CDR Mark L. Mitchell, and LCDR Thomas Bunker, both students at NPS conceived a logistics war game called PROLOG. PROLOG was initially developed as a board game that was played between two opposing sides with dice. In July of 1985 CDR Mitchell returned to NPS as an instructor and continued his work on the game. CDR Mitchell was able to develop the game into a computer-aided war game.

The current form of PROLOG is not the work of CDR Mitchell alone, but incorporated thesis work from five Operation Analysis Master's Degree students. It includes a surface ship combat simulation, an air combat simulation, and an air maintenance simulation. PROLOG has one battle scenario, in which a late 1980's battle group conducts air strikes and combats coastal craft controlled by the computer representing insurgents in "Etas Orango" (Mitchell 1988).

As effective as PROLOG is for its design goals, it has limitations that detract from its utility for modern logistics training. It was developed to model only the carrier battle group and has no amphibious ships or landing forces. Thus, it cannot be used to train officers in the concepts of OMFTS and SBL. Therefore PROLOG is not sufficient to train today's logistics students.

D. OBJECTIVE OF THESIS

This thesis lays the groundwork for and begins the implementation of an adaptable computer-based wargame that is designed to train logistics officers to support OMFTS and Sea-Based Logistics. The wargame that will ultimately be developed has been titled NAVLOGS.

NAVLOGS' objective is to not only function as a teaching aid like PROLOG, but also to introduce the logistician to the difficulties in maintaining the supply lines for an amphibious operation. NAVLOGS has the following improvements over PROLOG: an upgrade to the ship classes and weapon systems from 1988 to present day, a Graphic User Interface (GUI) to increase playability, a reduction in time to complete the game, and a MEU and ARG element to increase the complexity. NAVLOGS will read all unit data from an external database and include a Scenario Builder to allow for updates in units and tactics. These features will make NAVLOGS an adaptable training aid that will remain useful even in the face of future changes in assets or doctrine. NAVLOGS can be run on every major workstation or personal computer because of its implementation in Java, a platform independent language. A fully functioning NAVLOGS application is beyond the scope of this thesis alone. At its end state, NAVLOGS will combine elements implemented here with those of LT Troxell (Troxell 1999) and others.

The author designed and implemented several key components of NAVLOGS. The author's implementation of the Scenario Builder lets the user redesign scenarios to increase NAVLOGS flexibility to change. The newly created GUI accomplishes the interaction between the Scenario Builder and the user. The Scenario Builder includes coastline modeling and model-scaling applications that augment the Scenario Builder's adaptability. The coastline model and model scaler allow the simulation to use any digitized map as the source. This is a significant

advantage over previous simulations in which the maps had to be built into the program. While many systems use proprietary map information, NAVLOGS can use standard image formats for a map, such as gif or jpg. The model scaler enables the geo-referencing of any map image. Finally the author researched and supplied the database tables for the Marine Corps units that are used to configure the initial scenario. NAVLOGS' significant contribution over previous simulations is its flexibility to change.

The remainder of this thesis is organized as follows. Chapter II is a detailed description of the model design and the elements of NAVLOGS created by the author. In Chapter III an initial scenario is outlined, and its implementation is discussed. Finally, in Chapter IV the conclusion and future recommendations are given.

II. MODEL METHODOLOGY

This chapter provides the structure and explanation of the NAVLOGS components developed in this thesis. The general design concept will be discussed first, followed by the Scenario Builder and two of its integral parts, the model scaler and the coastline generator.

A. DESIGN

The general structure of NAVLOGS is modular in nature. An explanation of how the choice of Java as the computer language has influenced the modular design will be discussed first. NAVLOGS basic design structure is discussed next. Lastly this section explains the assumptions in the discrete event simulation model.

1. Programming Language

NAVLOGS was implemented in Java because it is Object-Oriented, platform independent, and supports component-based designs such as NAVLOGS. In Object-Oriented Programming (OOP) the program is broken down into entities (called "objects") that are responsible for their own related set of tasks. Each object consists of both data and methods that operate on the data. This allows a large project to be functionally decomposed. Another key OOP concept is encapsulation. By combining data and behavior into one package and hiding the implementation from the user of the data, OOP allows the loose coupling of classes. A loosely coupled model uses objects that provide information to each other and utilize that information to return results. The objects are not aware of how the other objects compute the data; the original object only gets back the final results. A loosely coupled model allows easy replacement of its objects. By loosely coupling the objects, future students will be able to improve them and plug

them back into the original simulation, without rewriting the code in its entirety. This is extremely important if NAVLOGS is to remain pertinent for more than a few years. Doctrine and equipment will change, so NAVLOGS must be adaptable to future changes.

Java's platform independence allows the same code that has been compiled on a Windows PC to be run on other systems such as Macintosh or UNIX. This becomes extremely advantageous when dealing with the diverse number of machines that are in DoD. The ultimate benefit gained through platform independence is that users are able to run the program on a machine with which they are comfortable. Any future operating systems will run NAVLOGS as long as there is an implementation of the Java Virtual Machine on it. The rapid growth of operating systems such as LINUX highlights the need for this ability to run on platforms yet to be implemented.

Another benefit gained by programming in Java was the ability to use previously designed classes such as Dr. Arnold Buss's Simkit. Simkit provides the underlying simulation engine for NAVLOGS. Its design, with that of NAVLOGS promotes modularity and loose coupling. The loose coupling of the modules in the simulation is essential to NAVLOGS design.

2. NAVLOGS' Components

Figure 1 illustrates NAVLOGS' modular design showing NAVLOGS' components. These include a database, the Scenario Builder, the discrete event simulation, which performs all the combat calculations, a GUI that will display the simulated equipment's location and status, and a controller that translates the user's inputs to the simulation and the simulation results to the user. LT Troxell designed the database, its interactive software, and the majority of the naval

units (Troxell 1999). The author designed and implemented the scenario builder, the ground combat elements, the amphibious ships, and the support craft for the ground elements.

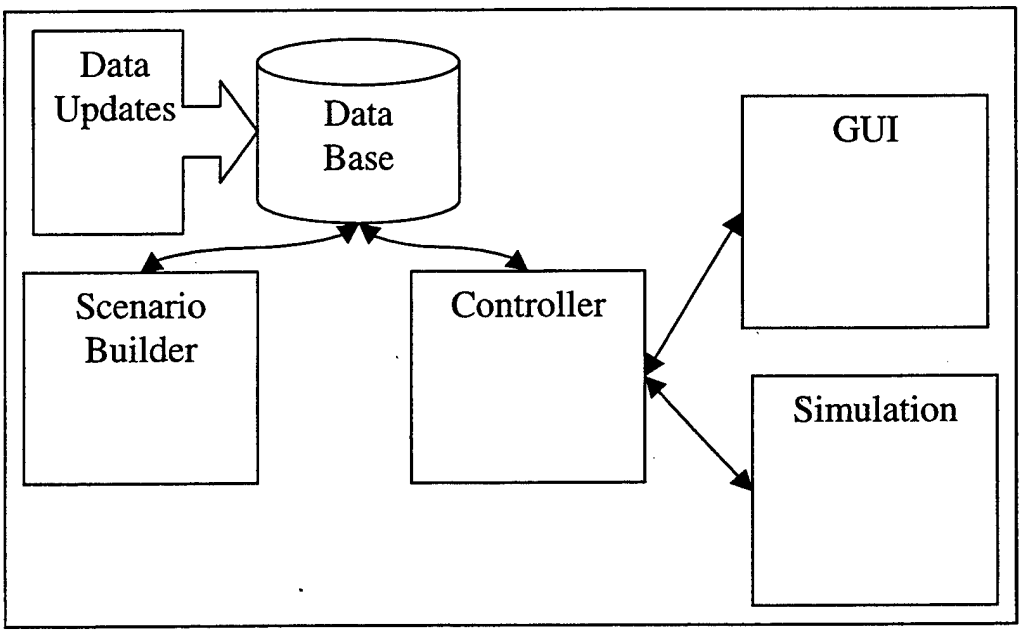


Figure 1 NAVLOGS Design Concept

Upon startup the controller queries the database and informs the simulation what units are available and provides it with the scenario data. The simulation then constructs the units needed, and begins the discrete event simulation. As the scenario unfolds, the controller begins sending unit locations and the geographical picture to the GUI, which displays them on the screen, allowing the user to interact with the simulation.

The separation of the GUI, the simulation, and the database by the controller is an important feature that is essential to its modularity. By including this separation these three components remain independent of each other, allowing changes to be made in any of them without necessarily affecting the other. The controller interprets the data provided by these three components and then passes them the information they require to run the wargame.

An innovation over PROLOG is its ability to utilize multiple scenarios. NAVLOGS includes a scenario builder, which allows a user to easily create additional scenarios for further training. NAVLOGS stores all unit and scenario information in the database to allow easy editing and the addition of new units. The characteristics of the Marine Expeditionary Unit (MEU) and its supporting Amphibious Readiness Group (ARG) as well as their aircraft and amphibious vehicles are stored in the database. The database allows these unit characteristics to be easily modified. For example, NAVLOGS database was designed with the use of unclassified data. If in the future a classified database is desired, all the user has to do is simply modify the data in the database. The characteristics of the units are influenced by the assumptions made in the model design.

3. Model Assumptions

Like every discrete event simulation, the one in NAVLOGS makes certain assumptions that abstract reality to a stylized model. However, the modular construction of NAVLOGS will allow future users to override these assumptions should they prove unacceptably limiting in future scenarios. The following are some of the logistics assumptions that were made in NAVLOGS.

Currently NAVLOGS only models fuel and ammunition expenditures, a decision based on the criticality of these two items as well as their weight. All other classes of supply are assumed incidental to fuel and ammunition. The major assumption being that if fuel and ammunition is able to be delivered other less heavy, items can be delivered with the primary resupply.

The NAVLOGS simulation uses a fuel consumption model designed by Dr. David Schrady et al. (Schrady 1996). These fuel curves are extremely accurate predictors for fuel usage that allow for a realistic implementation of fuel consumption based on speed. Accurate fuel consumption is needed to teach the user that by increasing speed he/she increases the frequency of the need for resupply.

The basic "unit" for a NAVLOGS exercise is the scenario. In NAVLOGS scenarios are created in the Scenario Builder, which is described in the next section.

B. SCENARIO BUILDER

The scenario builder allows users to graphically design their own scenarios easily using a sophisticated interface, permitting users to adjust scenarios to reflect current events. It allows them to place initial starting positions and dispositions for enemy and friendly forces. The graphical nature of the scenario builder allows fairly sophisticated models to be designed with little training on the system itself.

1. Maps and Model Scales

A scenario is built by first selecting a map to use. The map can be any gif or jpg image, as seen in Figure 2, which allows great flexibility.

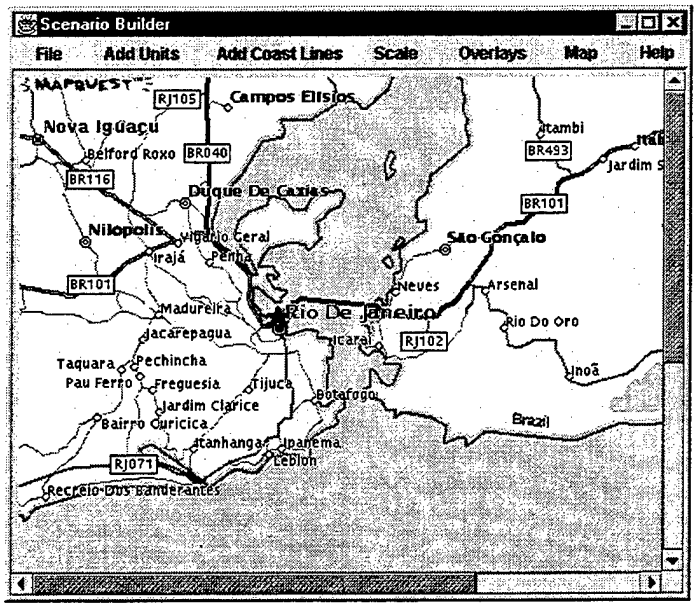


Figure 2 Scenario Builder with Map

The author implemented two ways to scale an image once it is chosen for use as a map. By selecting the scale pull down menu users are able to choose which method they prefer (See Figure 3).

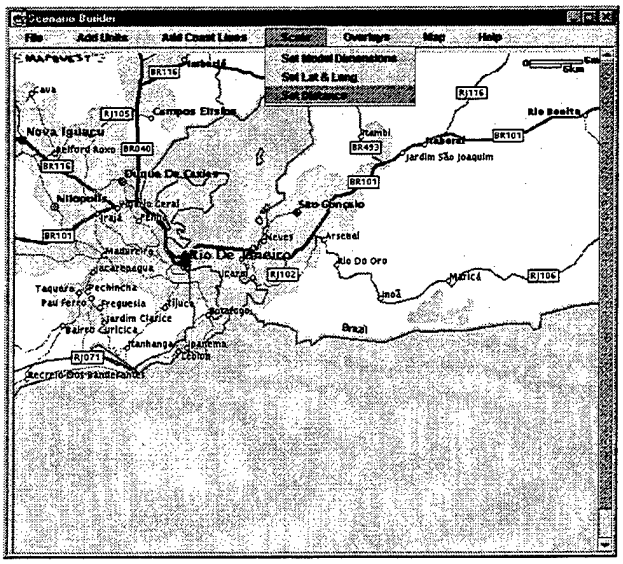


Figure 3 Scale Pull Down Menu

The first scaling method relies on the user providing two points of latitude and longitude. The image is assumed to be a Mercator projection, a common style of coastal charts. Since the lines of latitude are perpendicular to the lines of longitude in a Mercator projection the Scenario Builder extrapolates the boundaries of the model based on the data given. The Scenario Builder draws in the latitudinal and longitudinal lines to assist the user in unit placement, as seen in Figure 4. In the vicinity of the North or South Pole the map projection becomes distorted on a Mercator map. To overcome this distortion the set distance method permits the use of other types of maps.

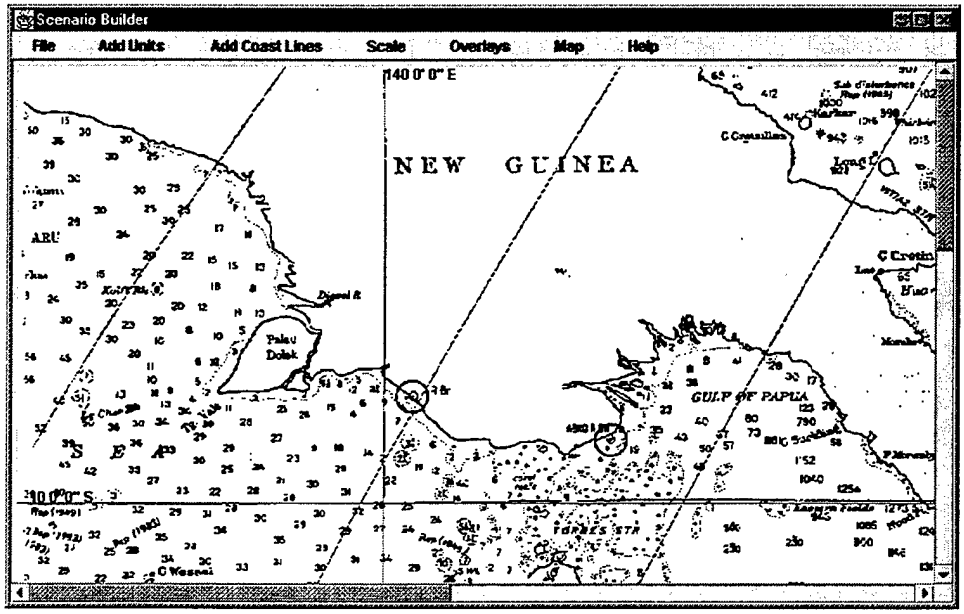


Figure 4 Scenario Builder with Latitude and Longitude Scale

The set distance method allows the user to utilize the map's own scale to set the model's scale. The user inputs two points again, but this time only has to enter the distance between them. This is useful for maps with a built-in scale as seen in Figure 5. The Scenario Builder is able to use this data to set the boundaries of the model. While this method does not generate a

geo-reference, it allows for a much wider range of maps than if several projections were modeled.

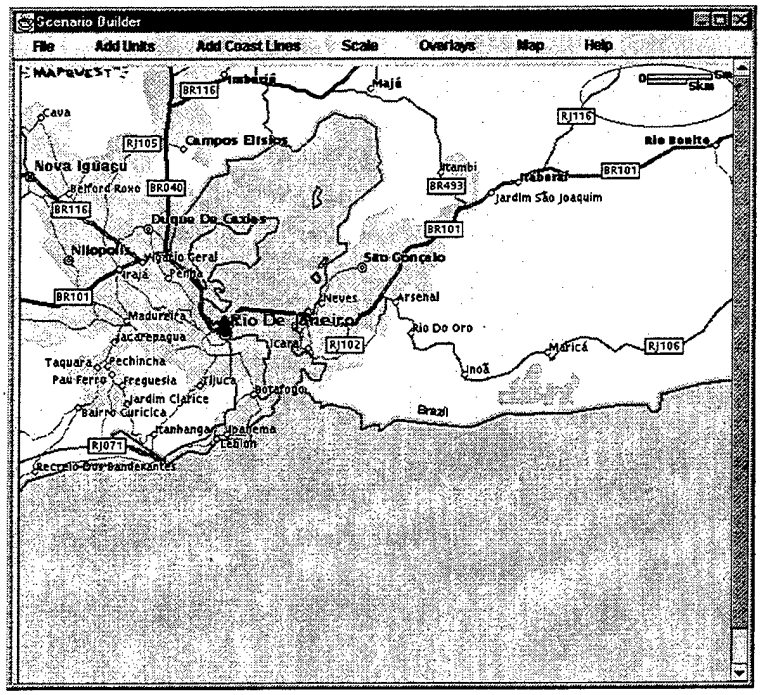


Figure 5 Setting Distance Scale

Once the model has a scale to reference, the user can graphically place new units on the map.

2. Units

The "Add Units" pull down menu is read from a text file that is independent of the GUI itself as seen in Figure 6. The text file provides the GUI with the separation of the unit types, the names of the units, and the graphic representation desired for the units. An example of this file type is found in Appendix A. To change the units in the pull down menu the user simply adjusts the file to match the units in the database.

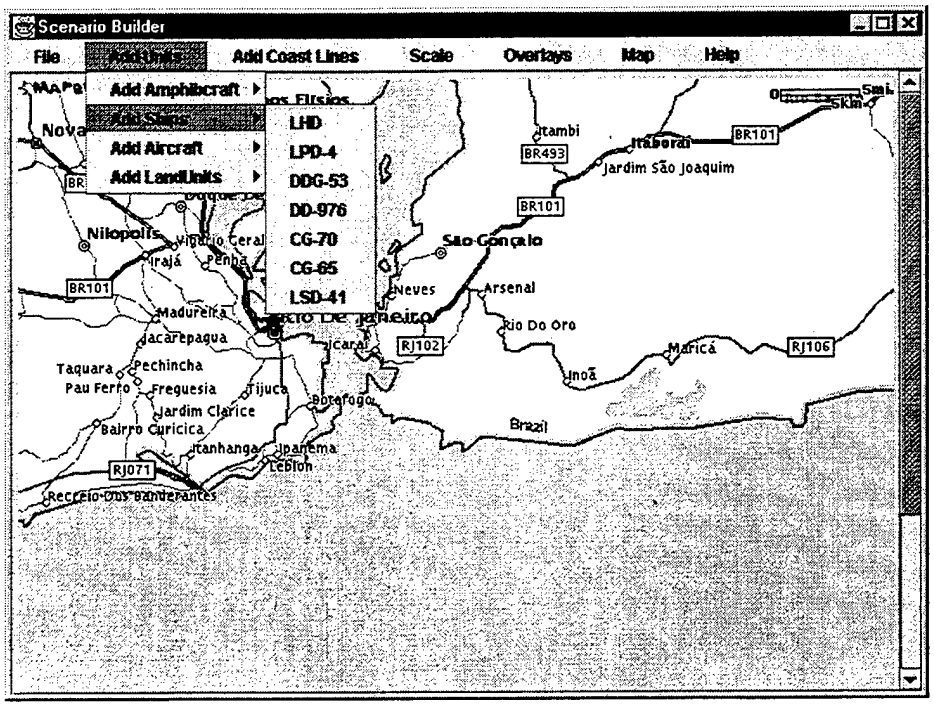


Figure 6 Add Units Pull Down Menu

In Figure 7 the user has chosen three ships from pull down menus and added them to the scenario with the mouse. The icons show that the units are surface ships, and are friendly. Additionally Figure 7 shows the "tool tip" displaying the ship's type. This appears when the user leaves the mouse pointer over top of any icon and helps identify the unit to the user.

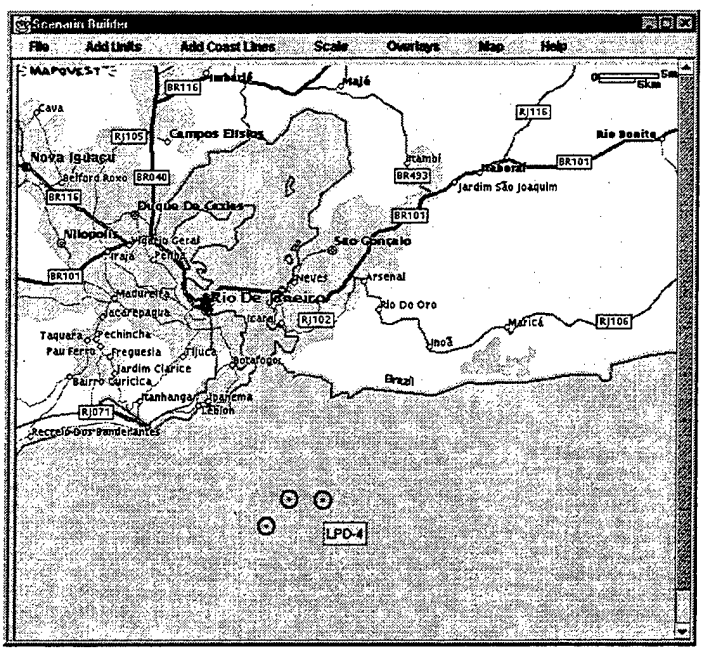


Figure 7 Adding Ships

Figure 8 shows that enemy land units have been placed using the same procedure.

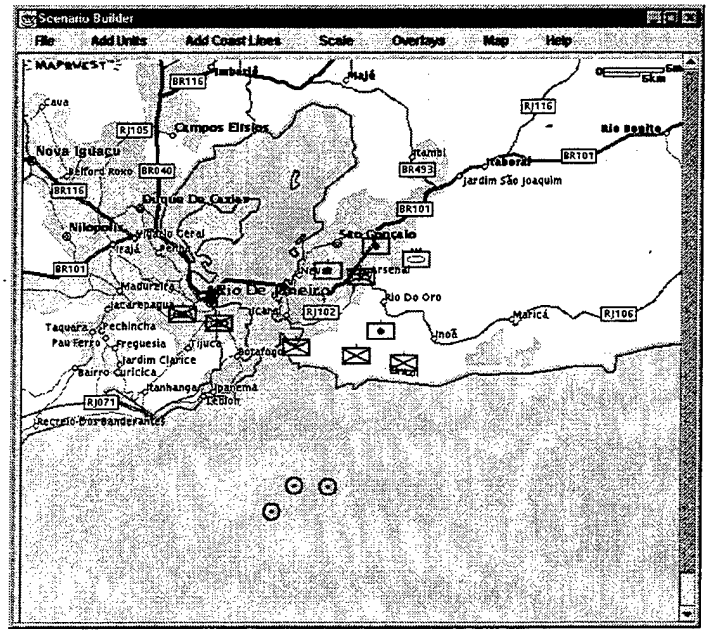


Figure 8 Adding Enemy Positions

Once the units and positions have been entered the user will be able to adjust the characteristics of the units to fit the individual scenario concept through the use of pop-up windows. (See Figure 9.) The data displayed in the pop up window is unit specific and varies from unit to unit. The database then records the model location of these units and their characteristics upon saving.

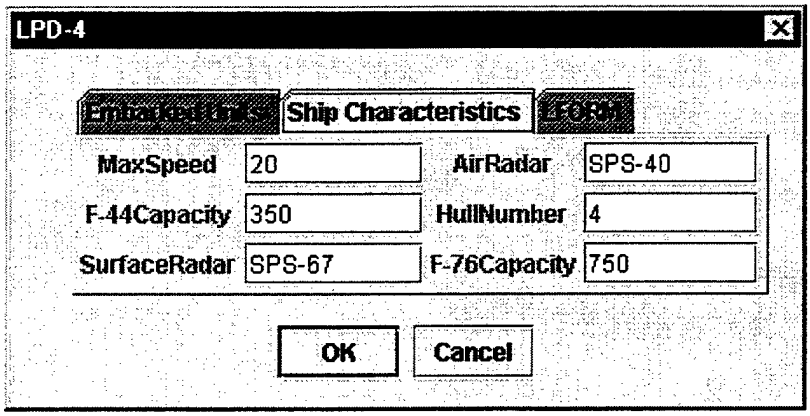


Figure 9 Sample Unit Characteristic Pop-Up Window

3. Coastlines

Since NAVLOGS is intended to allow for amphibious landings, a coastline that bars sea-based units from crossing onto land, and land-based units from crossing onto sea is required. More importantly most of the amphibious craft have a different max speed for land and sea. The model must be able to change this max speed based on the unit's location.

NAVLOGS provides the means to model coastlines as physical objects. The user in the Scenario Builder draws the coastlines on top of the map (See Figure 10). This allows the image that is being used as the map to be of any type, and does not require the image to provide the coastal data. In PROLOG there were no amphibious forces, and the model did not require an

extensive coastline; the programmers kept the ships from crossing the coastline by placing a minefield around the coastline that would destroy ships that came too close.

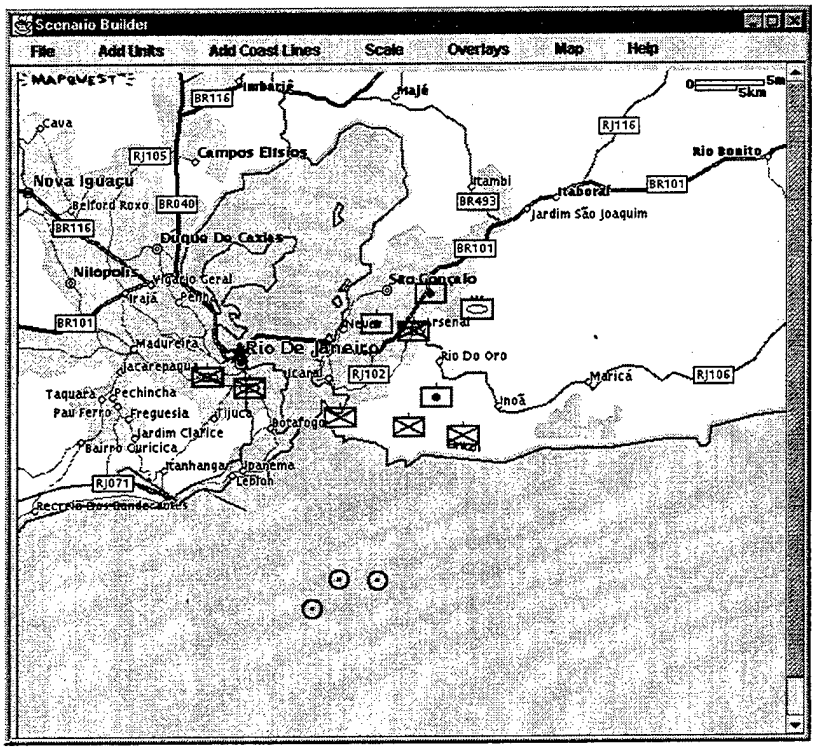


Figure 10 Drawing Coastlines

The coastal model in NAVLOGS is stored as a vector of LineSegment objects. The LineSegment class adds support for a piecewise linear model of a coastline. Instances of LineSegment store the endpoints together with a normal vector to the segment. The normal vector points toward the landside of the coastline.

A LineSegment object determines when a unit is attempting to cross a coastline. To accomplish this purpose LineSegment has two specific methods: the timeToIntersect(), and the doTheyIntersect(); these compare line segments with a unit's trajectory and determines if the unit intersects it, and at what time. To illustrate, in Figure 11 points (x_0, y_0) and (x_1, y_1) represent a

line segment with a normal vector **n**. Points (x_2, y_2) and (x_4, y_4) represent the starting point and end point of a moving land unit with a velocity of vector **a**. NAVLOGS checks all new movement commands against its vector of coastlines to determine if the path of a moving unit crosses the coastline using the following equations:

$$\mathbf{b} = (x_2 - x_0, y_2 - y_0)$$

$$t = \frac{\mathbf{n} \bullet (-\mathbf{b})}{\mathbf{n} \bullet \mathbf{a}}$$

where t represents the time required to reach the coastline. If t is positive, the point $(x_3,y_3) = (x_2,y_2) + \mathbf{a}*t$ is the intersection point. Once the point (x_3, y_3) is determined the simulation then checks to see if it is between the two endpoints of the line segment. If $MIN(x_0, x_1) \le x_3 \le MAX(x_0, x_1)$, then (x_3, y_3) is on the line segment. If (x_3, y_3) is determined to be on the line segment, then the simulation stops the unit's movement at (x_3, y_3), since land vehicle cannot enter the water.

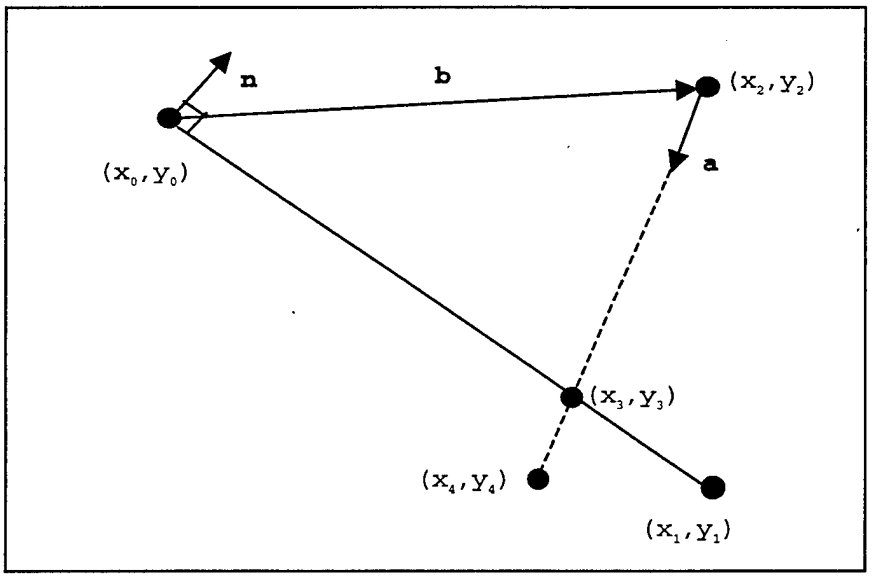


Figure 11 Line Segment Intersection

If the unit is an amphibious vehicle, then it can proceed through point (x_3, y_3) to point (x_4, y_4), albeit at its speed in the water. All moves originating on a coastline are checked against the normal of that line segment. In Figure 12 a unit is on the segment at (x_0, y_0) and initiating a move to point (x_2, y_2). In this case, $\mathbf{n} \bullet \mathbf{b} \geq 0$, so (x_2, y_2) is on the landside of the coastline. The simulation now allows the move if the moving unit is a land unit, disallows the move if the unit is a sea unit, or sets the unit's max speed to its land speed if the unit is amphibious.

Taken together, the line segments represent boundaries that constrain the movement of land and sea units. The above computations are the basis for implementing these constraints in NAVLOGS' discrete event simulation.

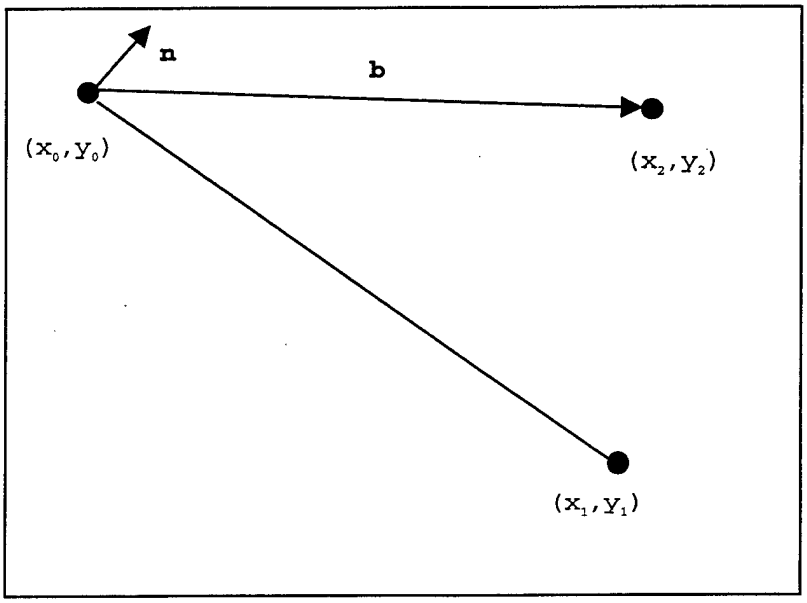


Figure 12 Movement From a Line Segment

Once the physical envrionment and the unit locations are in position the user can then create a script that schedules future events on the simulation's event list. The script allows the the user to schedule troop movements, reinforcement arrival times, and victory conditions. The

model controller will interpret the event list and then pass the information to the simulation's discrete event set. Thus without intricate knowledge of the program's code and without reprogramming the model, the user is able to change scenarios, unit lists, and concepts of operations in NAVLOGS.

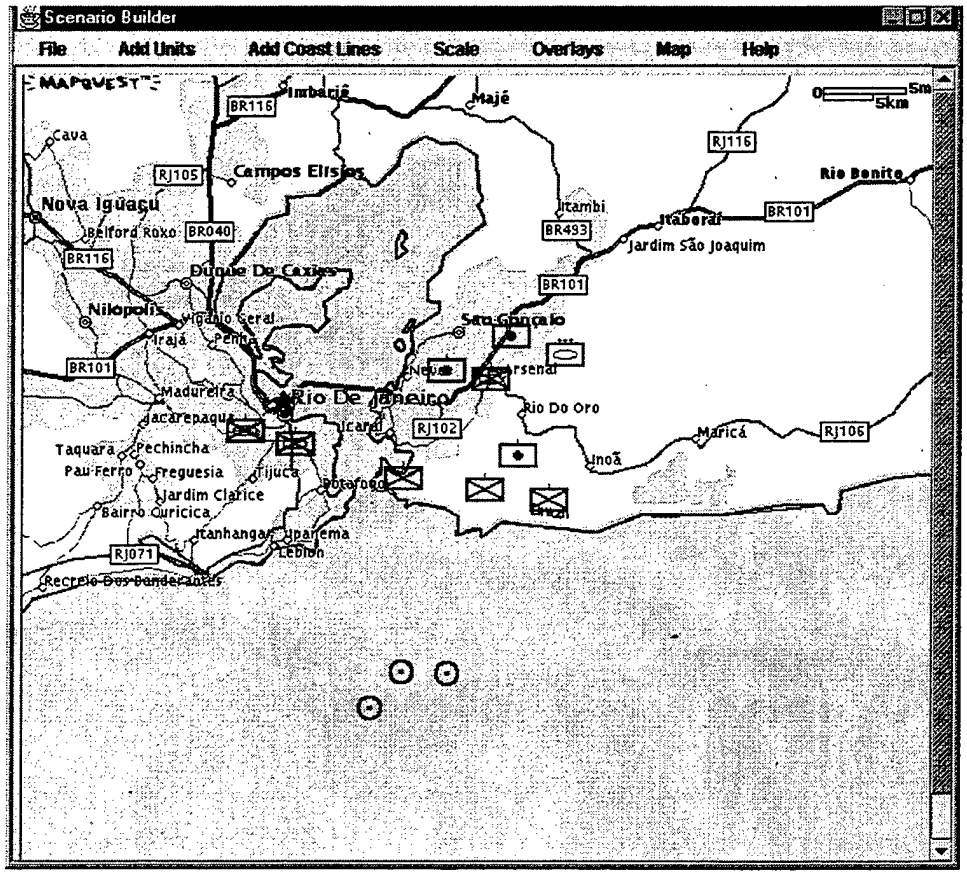


Figure 13 Finished Scenario

C. SUMMARY

The OOP aspect of Java was integral to NAVLOGS modular design. It is through its modularity that NAVLOGS gains its adaptability to change and is able to utilize the author's

Scenario Builder. Without this NAVLOGS would be constrained to a single immutable scenario like PROLOG.

The Scenario Builder is designed for the user to be able to customize the model to fit the training objectives. This enables the model to be removed from the programmer's hands and placed into that of the logistician or war-fighter. NAVLOGS is thus more useful because now the end user can redesign the scenario without necessarily being an expert computer programmer.

The Scenario Builder's ability to use any computerized image as a map hinges on its two key components. The model scaler translates the image's pixel location to a scale the simulation can use, allowing an accurate portrayal of unit speeds. The coastline model enables the user to enter the coastline as a series of line segments, freeing the simulation from having to rely on the complex representations generated by the Defense Mapping Agency or other similar institutions.

The generation of scenarios is one of NAVLOGS' most important abilities. The next section will discuss the initial scenario, thus showing by example what the Scenario Builder can actually accomplish.

III. INITIAL SCENARIO

The initial scenario included with NAVLOGS is designed as a logistics trainer as well as a template for future users. The units included are taken from a typical MEU load out, and should be reusable for future scenario construction. The scenario itself is discussed following the description of the units. The outline of the events and adversarial actions in the scenario is included to give the reader a description of the level of intricacy that can be included in a NAVLOGS scenario.

A. UNITS

The ARG element of the simulation consists of a *Wasp* class helicopter carrier (LHD), an *Austin* class Landing Platform Dock (LPD), and a *Whidbey Island* class Landing Ship Dock (LSD). These ships are grouped to support the Marine units that are typically included aboard during a deployment. (See table 1) The ships are capable of refueling the embarked aircraft and vehicles. They will provide Landing Force Munitions (LFORM) to the Marines ashore via LCAC or airlift. The LHD primarily supports the Air Combat Element (ACE), while the LPD and LSD will provide support for the Ground Combat Element (GCE), and the Combat Service Support Element (CSSE). The Command Element (CE) is not included, since the human player will provide that role. The ACE will provide Close Air Support (CAS). These units are modeled using today's equipment. Of course, as discussed in Chapter II, NAVLOGS will be easily modified to incorporate future assets.

ClassName	HullNumber	SurfaceRadar	AirRadar	MaxSpeed	F-76Capacity	F-44Capacity	b0	b1	b2
Amphib	LPD-4	SPS-67	SPS-40	20	750	350	-1124.43	1566.79	95.46
Amphib	LSD-41	SPS-67	SPS-49	22	597	53	-32454.8	32693.5	2.86
Amphib	LHD-1	SPS-67	SPS-48	22	1850	367	-700.81	2039.41	78.21

Table 1 Partial Amphibious Ship Data

The ACE has combat aircraft as well as support aircraft. Among these units are the 12 MV-22 Osprey class tilt-rotor aircraft. The Osprey is designed to be the workhorse for the Marine Corps, replacing the CH-46, and is the only future aircraft used in the model. The ACE also has 4 CH-53s for troop insertion and additional support. There are 3 UH-1N Huey aircraft, which can be used for command and control or aerial spotting. The combat aircraft of the ACE are the 6 AV-8B Harrier II VTOL attack jets. These will perform the majority of the close air support (CAS) missions for the Marines on the ground. The 4 AH-1W Cobra attack helicopters will provide back up air cover in a similar role to the Harriers. With the possibility of air support from the carrier battle group designed in LT Troxell's default scenario the aviation aspect of NAVLOGS will be quite well-rounded.

The GCE consists mainly of an Advanced Assault Amphibian Vehicle (AAAV) platoon of 13 AAAVs. Moving directly from the ships to shore and overland to the objective, these vehicles provide the most protected way to deliver troops ashore. The GCE also has 3 rifle companies of approximately 150 Marines each. These infantry companies require either air transportation or sea-born transportation to make it ashore, but since they are foot mobile they do not require fuel. For rapid movement ashore the Marines have a heavy weapons company consisting of 24 armored HMMWVs with Tow missile mounts and 81-mm grenade launchers. The GCE also has a Light Armored Vehicle (LAV) reconnaissance platoon with 6 LAVs. The reconnaissance platoon acts as the scouts for the ground units and will be able to move in rapidly

to scout an area and fall back to a secure position. For heavy firepower the GCE has an artillery battery consisting of 6 M198 Howitzers. The Howitzers require 5-ton trucks for movement, as well as for transport of ammunition. The artillery battery will be one of the most logistic intensive units used. Finally the GCE has 4 M1A1 tanks which will be available should they be necessary. (See table 2.)

javaClass	name	currentSpeed	maxSpeed	waterSpeed	GPH	length	width	troopCap	weight
Semi	MK48	0	50	0	16.66	238.5	96	0	25300
Trailer	MK14	0	0	0	0	238.5	96	0	16000
Trailer	MK15	0	0	0	0	240	96	0	26000
Trailer	MK17	0	0	0	0	240	96	0	22650
Truck	AMB	0	50	0	1.7	202	85	4	7168
Truck	M92	0	50	0	11.5	332	97	12	22700
Truck	M927	0	50	0	11.5	410.4	98	12	25620
Truck	DMPTRK	0	50	0	11.5	294.5	97	0	24960
Truck	M1044	0	50	0	1.7	185	85	4	6104
Truck	M1038	0	50	0	1.7	185	85	4	5140
Truck	5TON	0	50	0	13	362	97.4	12	36910
Truck	5TONWRKR	0	50	0	13	362	97.4	0	36910
Truck	M88A1	0	25	0	20	338	144.5	12	139000
FuelContainer	SIXCON	0	0	0	0	96	80	0	4000
AmphibVehicle	AAAVC7	0	45	30	9	353	174	18	70924
AmphibVehicle	AAAVP7	0	45	30	9	353	174	18	70924
AmphibVehicle	AAAVR7	0	45	30	9	353	174	18	70924
AmphibVehicle	LAV25MM	0	50	3	7	224	89	2	21800
AmphibVehicle	LAVAT	0	50	3	7	224	89	2	21800
AmphibVehicle	LAVL	0	50	3	7	224	89	2	21800
AmphibVehicle	LCAC	0	50	50	700	972	524	25	333200
Tank	M1A1	0	41	0	17.3	356	144	0	126000

Table 2 Partial Land Vehicle Data

The Enemy units are lightly armed guerilla units. Their preferred method of attack will be from concealment and they will fight a harassing battle. The purpose of these tactics in this scenario is to erode the Marines' logistics as rapidly as possible with little actual damage to their

units, thus maximizing resupply requirements. The enemy units possess some mechanized infantry consisting largely of older Soviet style equipment, and weapons. Additionally they possess shoulder fired Surface to Air Missiles (SAMs) that will hinder any aerial assault launched by the MEU. This effectively prolongs the scenario and requires resupply for the units, in accordance with the goals of the game.

B. SCENARIO

The scenario itself will be coordinated with LT Troxell's naval scenario (Troxell 1999). Some time after the beginning of the naval scenario the MEU will receive word that they are required to evacuate the US embassy. They will be advised of the SAM threat, and will be authorized to launch a land assault over the beach. Once ashore they are to proceed inland to the embassy and secure the area so the helicopters can land safely to evacuate the US nationals. Once on land the US forces will receive harassing fire from the enemy units, which will slow the rate of movement and cause the Marines to expend ammunition. After reaching the embassy and securing the area, the MEU will be informed that several US nationals are trapped in a factory outside of the town. This will begin the second phase of the operation in which the Marines proceed through the city and seize the factory in question. Again they will come under attack, and be required to fire upon the enemy forces.

The scenario should last long enough to force the player to rearm and refuel the forces several times. This should exercise the students' ability to plan an OMFTS resupply operation and demonstrate the difficulties involved. The scenario is just an initial scenario; it can be rewritten easily should it prove too hard or easy. The instructor and the students will be able to

alter the scenario or unit characteristics. For example: Suppose that in the future the AAAV (C7)'s troop capacity increases from 18 to 20. The user would merely need to modify the database entry for AAAV (C7) from Table 3 to Table 4.

javaClass	name	maxSpeed	waterSpeed	maxF44	GPH	length	width	troopCap	weight
AmphibVehicle	AAAVC7	45	30	400	9	353	174	18	70924

Table 3 Sample Database Entry

javaClass	name	maxSpeed	waterSpeed	maxF44	GPH	length	width	troopCap	weight
AmphibVehicle	AAAVC7	45	30	400	9	353	174	20	70924

Table 4 Sample Database Entry Modified

Once the database is modified the user only needs to save it and restart the program. There is no need for recompilation or modification of code in any form. Similarly by creating a new scenario with the Scenario Builder, the simulation can be changed drastically without modifying the underlying program.

The initial scenario described above is intended to not only instruct players in OMFTS logistics difficulties, but to act as a guideline for future scenario design. An entire library of scenarios could be built and stored in databases allowing the user to practice different logistic plans. This is one of the fundamental goals of NAVLOGS.

C. DISCUSSION

The initial scenario exposes the user to the difficulties faced by OMFTS operations. The simulation in NAVLOGS interacts with the user via the GUI allowing the user to make Task Force Commander level decisions. These decisions affect the fuel and ammunition consumption

of the sea and land based units. As the supply levels diminish the user must then schedule resupply via air or sea. Demonstrating the challenges of OMFTS' concept is the primary goal of the initial scenario.

While OMFTS is concentrated on in the initial scenario future scenarios could exercise other logistics concerns. By having scenarios separate from its internal code NAVLOGS can be utilized to exercise logistical plans not yet conceived. This is a major improvement over PROLOG.

IV. CONCLUSIONS

Drastic changes in the political balance of the world in the past decade have had a significant impact on the United States military. The disappearance of a clear adversary in the form of the Soviet Union and a rise in the economic influence of the global market leave the United States to face uncertainty in future conflicts which will be limited in scope and regional in nature. To face the future of combat the U. S. Marine Corps has developed OMFTS. This concept of operations relies on the Navy to provide logistical support to the Marines ashore in the form of a sea-base. Success in OMFTS based missions requires enhanced cohesiveness between Navy and Marine Corps logisticians. The computer training aid NAVLOGS encourages forward thinking in this area of interest.

NAVLOGS is written in JAVA, a platform independent object-oriented computer language. Platform independence is important to ensure that the program will be able to be run on future operating systems. JAVA's object-oriented capabilities facilitate NAVLOGS' loosely coupled nature. NAVLOGS exploits the model-view controller architecture for flexibility and extensibility. The three major components of NAVLOGS, the database, GUI, and discrete event simulation, interact through the use of the controller which allows replacement of individual components without affecting the others.

The Scenario Builder developed in this thesis allows the user to redesign scenarios as well as create new ones. This ability can be used to model current, real world situations and help train the user to foresee logistics shortfalls. The graphical interface of the Scenario Builder allows users with little or no programming skills to develop a scenario, thus putting the tool in the hands of the logistician and war fighter. The user is able to choose any digitized map for the

scenario's source and create a scale for the map via a pull-down menu. The units are then placed on the map by selecting and dragging with the mouse. Coastlines can be drawn on the map allowing the simulation to model amphibious campaigns. With the coastlines represented properly the simulation can be used to represent both sea units and land units realistically. The Scenario Builder will enable NAVLOGS to remain relevant to future policies and conflicts, and it ensures the model will be applicable for future users.

NAVLOGS includes an initial scenario designed to teach the difficulties of SBL and OMFTS. The database for the initial scenario has the units included in a typical MEU and is reusable in future scenarios. The database's reusability allows the initial scenario to act as a template from which future scenarios can be modeled.

NAVLOGS is designed to be a robust self-contained computer-aided wargame. Its goals are to teach future Navy and Marine Corps logisticians to consider the challenges that will be faced in resupplying an OMFTS operation. While this thesis alone has not fully implemented the NAVLOGS design, the development has laid a strong foundation on which future efforts may build.

A. FUTURE WORK

As mentioned above, this thesis has not taken NAVLOGS to its end state. The integration of the scenario builder and the database requires fine-tuning before the two components function seamlessly. The GUI for the actual game portion of NAVLOGS requires more work before it is usable, and a save game method has not been completely developed. Future students could complete this project.

A definitive combat model for NAVLOGS is alone, a thesis level project. The current combat model lacks the fidelity required to utilize NAVLOGS as a tactics trainer. With a properly upgraded combat model, NAVLOGS could prove versatile enough to function as both a logistics trainer, and a tactics trainer. This could prove extremely beneficial in getting the war-fighter to consider the logistics tail prior to over extending oneself.

The ship fuel curve models designed by Dr. Schrady enhance the ability of NAVLOGS to model realistic logistic needs. Future students could model vehicle specific, land-based fuel consumption curves. A model for land vehicles similar to the fuel curves generated by Dr. Schrady would not only increase the reality level of the game, but would provide a valuable planning tool for operations that have severe time constraints.

With these improvements NAVLOGS can become an extremely sophisticated training aid that could prove invaluable to future logistics students.

APPENDIX A. UNITS.INI FILE

The units.ini file below lists the units available to the Scenario Builder grouped according to their classification, and includes the file reference to the graphic that represents the unit in NAVLOGS. The Scenario Builder reads this file, and creates the units pull down menu directly from it. This allows the addition of new units or even entire unit classes without having to rewrite the Scenario Builder.

```
[Ships]
DDG-53 = ships/Friendlysurf.jpg
CG-65 = ships/Friendlysurf.jpg
DD-976 = ships/Friendlysurf.jpg
LPD-4 = ships/Friendlysurf.jpg
LSD-41 = ships/Friendlysurf.jpg
LHD = ships/Friendlysurf.jpg


[LandUnits]
HMMWV = ships/mechinf.jpg
5-Ton = ships/mechinf.jpg
M1A1  = ships/armor.jpg


[Aircraft]
MV-22 = ships/Friendlyair.jpg
F-14 = ships/Friendlyair.jpg
Harrier = ships/Friendlyair.jpg
LCAC = ships/LCAC.jpg


[Amphibcraft]
AAAV = ships/aaav.jpg
LAV = ships/lav.jpg
```

APPENDIX B. LINESEGMENT CLASS

Described above the LineSegment class is the element used to store the coastlines in NAVLOGS. This class allows queries against the segments of the coastlines by moving units to determine if and when intersection of the coastline occurs. This accurately models a coastline obstacle in a discrete event methodology.

```
package navlogs.smd;
/**
 * John R. Sterba <BR>
 * LineSegment<BR>
 * April 25, 1999
 * <P>
 * Comments.. LineSegmant is a class which records a line segment
 * in two dimensional space, and determines its parallel, and
 * normal vector.  It is then able to determine if it intersects
 * another line segment, and when that intersection will occur.
**/
import simkit.smd.*;
import simkit.data.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class LineSegment {

//instance variables
   Coordinate point1;
   Coordinate point2;
   Coordinate normalVector;
   Coordinate parallelVector;

//constructors

   public LineSegment(String x1, String y1, String x2, String y2) {
      this(new Double(x1).doubleValue(), new Double(y1).doubleValue(),
           new Double(x2).doubleValue(), new Double(y2).doubleValue());
   }

   public LineSegment(double x1, double y1, double x2, double y2) {
      point1 = new Coordinate(x1,y1);
      point2 = new Coordinate(x2,y2);
      parallelVector = new Coordinate(point2).decrementBy(point1);
      if (parallelVector.getNorm() > 0) {
         parallelVector = new Coordinate(parallelVector).scalarMultiply(
                1/parallelVector.getNorm());
      }
      normalVector = new Coordinate(parallelVector.getYCoord(),(
```

```
                -1 * parallelVector.getXCoord()));
    }

    public LineSegment(Coordinate a, Coordinate b) {
       this(a.getXCoord(), a.getYCoord(), b.getXCoord(), b.getYCoord());
    }


//instance methods

    public Coordinate getPoint1(){return new Coordinate(point1);}

    public Coordinate getPoint2(){return new Coordinate(point2);}

    protected void setParallelVector(Coordinate a) {
       double x = a.getXCoord()/parallelVector.getXCoord();
       if ((x*parallelVector.getYCoord()) == a.getYCoord()) {
          parallelVector = new Coordinate(a);
       }
       else {
          System.out.println("Wrong vector for given line segment.");
       }
    }

    public Coordinate getParallelVector(){
       return new Coordinate(parallelVector);
    }

    public Coordinate getNormalVector(){
       return new Coordinate(normalVector);
    }

/**
  * timeToIntersect determines the scalar t that must be
  * applied to the formula x + a*t where x is a point in the plane
  * of the reference line segment and a is a parallel direction
  * vector in order for the line defined by x and a to intersect
  * the line that is represented by the reference LineSegment.
  * Both x and a are given as Coordinates.
**/
    public double timeToIntersect(Coordinate x, Coordinate a)  {
       Coordinate n = new Coordinate(getNormalVector());
       double n1 = n.getXCoord();
       double n2 = n.getYCoord();
       Coordinate p = new Coordinate(getPoint1());
       double p1 = p.getXCoord();
       double p2 = p.getYCoord();
       double x1 = x.getXCoord();
       double x2 = x.getYCoord();
       double a1 = a.getXCoord();
       double a2 = a.getYCoord();
       return ((n1*(p1-x1) + n2*(p2-x2))/(n1*a1 + n2*a2));
    }

/**
  * doTheyIntersect determines whether the current LineSegment
  * intersects another given the two endpoints (x, y) and a
  * parallel vector (a).
```

```
**/
   public boolean doTheyIntersect(Coordinate x, Coordinate y, Coordinate a) {
      Coordinate n = new Coordinate(getNormalVector());
      double t;
      Coordinate xPoint;
      Coordinate p;
      Coordinate q;
      double x1;
      double y1;
      double p1;
      double q1;
      double xPoint1;
      if (n.dotBy(a) != 0) {
         t = timeToIntersect(x,a);
         if  (t >= 0) {
            p = new Coordinate(getPoint1());
            q = new Coordinate(getPoint2());
            x1 = x.getXCoord();
            y1 = y.getXCoord();
            p1 = p.getXCoord();
            q1 = q.getXCoord();
            xPoint = new Coordinate(x).incrementBy(new
Coordinate(a).scalarMultiply(t));
            xPoint1 = xPoint.getXCoord();
            if (xPoint1 >= Math.min(x1,y1) && xPoint1 <= Math.max(x1,y1)) {
               if (xPoint1 >= Math.min(p1,q1) && xPoint1 <= Math.max(p1,q1)){
                  return true;
               }
            }
         }
      }
      return false;
   }

   public boolean doTheyIntersect(LineSegment b) {
     return doTheyIntersect(b.getPoint1(), b.getPoint2(),
b.getParallelVector());
   }


   public String toString() {
      return "From " + point1 + " to " + point2 + "\nParallel to " +
            parallelVector + "\nNormal to " +normalVector ;
   }

//main method

   public static void main (String[] args) {
      LineSegment a;
      LineSegment b;
      String c1 = new String("0");
      String d1 = new String("0");
      String e = new String("10");
      String f = new String("10");
      a = new LineSegment(c1,d1,e,f);
      b = new LineSegment(0,5,5,5);
      System.out.println(a);
      System.out.println(b);
```

```
        System.out.println(a.timeToIntersect(b.getPoint1(),
    b.getParallelVector().scalarMultiply(5)));
        System.out.println(a.doTheyIntersect(b));
        RandomStream number = new RandomStream(8313939603L);
        int k;
        int j;
        int l;
        int m;
        int k1;
        int j1;
        int l1;
        int m1;
        LineSegment c;
        LineSegment d;
        System.out.println();
        for (int i = 0; i < 10; i++) {
           j = number.uniformI(0, 100);
           k = number.uniformI(0, 100);
           l = number.uniformI(0, 100);
           m = number.uniformI(0, 100);
           j1 = number.uniformI(0, 100);
           k1 = number.uniformI(0, 100);
           l1 = number.uniformI(0, 100);
           m1 = number.uniformI(0, 100);
           c = new LineSegment((double) j, (double) k, (double) l, (double) m);
           d = new LineSegment((double) j1, (double) k1, (double) l1, (double)
    m1);
           System.out.println(i);
           System.out.println(c);
           System.out.println(d);
           System.out.println(c.timeToIntersect(d.getPoint1(),
    d.getParallelVector().scalarMultiply(5)));
           System.out.println(c.doTheyIntersect(d));
           System.out.println();
        }
     }
    }
```

APPENDIX C. SCENARIO BUILDER GUI CLASS

The ScenarioBuilderGUI class displayed below shows the reader the code that allows the Scenario Builder the flexibility in adding new units. The units pull down menu is read from the units.ini file (Appendix A) and is not included in the code below.

```
package newt;
/**
 * John R. Sterba <BR>
 * ScenarioBuilderGUI<BR>
 * May 15, 1999
 * <P>
 * Comments.. ScenarioBuilderGUI is a class which creates and
 * displays the Graphic User Interface for the Scenario Builder
 * of NAVLOGS.
**/
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.lang.*;
import simkit.util.*;
import java.io.File;

public class ScenarioBuilderGUI extends JFrame {

//instance variables
   private String title;
   private DrawLines2 p2;
   private LatLongDrawPanel scalePanel;
   private Dimension model;
   private Action newCoast;
   private Action endCoast;
   private Action endIsle;
   private Action mapAction;
   private Action removeCoast;
   private JFrame f;
   private JTextField xField;
   private JTextField yField;
   private WindowCloser dimWC;
   private ImageIcon map;
   private JPanel pan;
   private DrawImage pan2;
   private JScrollPane p;
   private JLayeredPane lp;
   private JLabel l;
   private JLabel l2;
   private JLayeredPane pane;
   private INIFileProperties unitINI;

//constructors

   public ScenarioBuilderGUI(String t, String unitFile) {
```

```
        super(t);
        pan = new JPanel(true);
        pan2 = new DrawImage();
        title = t;
        unitINI = new INIFileProperties();
        unitINI.load(unitFile);
        p2 = new DrawLines2(title);
        scalePanel = new LatLongDrawPanel(title);
        AppCloser closer = new AppCloser(this);
        p = new JScrollPane(
              JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
              JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        this.getContentPane().add(p);
        lp = new JLayeredPane();
//creates the menu bar and populates it with menu items
        JMenuBar menu = new JMenuBar();

        JMenu fileMenu = new JMenu("File");
        menu.add(fileMenu);
        JMenuItem newItem = new JMenuItem("New Scenario");
        JMenuItem openItem = new JMenuItem("Open Scenario");
        JMenuItem closeItem = new JMenuItem("Close Scenario");
        fileMenu.add(newItem);
        fileMenu.add(openItem);
        fileMenu.add(closeItem);
        fileMenu.addSeparator();
        JMenuItem saveItem = new JMenuItem("Save Scenario");
        JMenuItem saveAsItem = new JMenuItem("Save Scenario As");
        fileMenu.add(saveItem);
        fileMenu.add(saveAsItem);
        fileMenu.addSeparator();
        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(new GenericAction(closer, "close"));
        fileMenu.add(exitItem);

        JMenu unitsMenu = new JMenu("Add Units");
        JMenu classMenu;
        JMenuItem unitItem;
        menu.add(unitsMenu);
        Properties prop = null;
        int itemNumber = -1;
 //lists the available units as read form the unit.ini file
        for (Enumeration e = unitINI.keys(); e.hasMoreElements();) {
           Object key = e.nextElement();
           classMenu = new JMenu("Add " + key.toString());
           unitsMenu.add(classMenu);
           try {
              prop = (Properties) unitINI.get(key);
              for (Enumeration f = prop.keys(); f.hasMoreElements();) {
                 itemNumber++;
                 Object key2 = f.nextElement();
                 unitItem = new JMenuItem(key2.toString());
                 unitItem.addActionListener(new EventCommandAction(this,
"addUnit"));
                 classMenu.add(unitItem);
              }
           }
           catch (ClassCastException ex) {}
           catch (NullPointerException ex) {}
```

```
        }

        JMenu coastMenu = new JMenu("Add Coast Lines");
        menu.add(coastMenu);
        newCoast = new GenericAction(this, "addNewCoastLine");
        coastMenu.add(newCoast);
        endCoast = new GenericAction(this, "endCoastLine");
        removeCoast = new GenericAction(this, "removeLastCoast");
        removeCoast.setEnabled(false);
        coastMenu.add(removeCoast);
        endCoast.setEnabled(false);
        coastMenu.add(endCoast);
        endIsle = new GenericAction(this, "endCoastAsIsleOrLake");
        endIsle.setEnabled(false);
        coastMenu.add(endIsle);

        JMenu scaleMenu = new JMenu("Scale");
        menu.add(scaleMenu);
        JMenuItem dimItem = new JMenuItem("Set Model Dimensions");
        JMenuItem latLongItem = new JMenuItem("Set Lat & Long");
        latLongItem.addActionListener(new GenericAction(this, "setLatLong"));
        JMenuItem distItem = new JMenuItem("Set Distance");
        scaleMenu.add(dimItem);
        scaleMenu.add(latLongItem);
        scaleMenu.add(distItem);

        JMenu overlayMenu = new JMenu("Overlays");
        menu.add(overlayMenu);
        ButtonGroup coastbg = new ButtonGroup();
        ButtonGroup scalebg = new ButtonGroup();
        ButtonGroup unitbg = new ButtonGroup();
        ButtonGroup mapbg = new ButtonGroup();
        JRadioButtonMenuItem showCoastItem = new JRadioButtonMenuItem("Show
Coast Overlay");
        showCoastItem.addActionListener(new GenericAction(this, "showCoast"));
        JRadioButtonMenuItem hideCoastItem = new JRadioButtonMenuItem("Hide
Coast Overlay");
        hideCoastItem.addActionListener(new GenericAction(this, "hideCoast"));
        JRadioButtonMenuItem showScaleItem = new JRadioButtonMenuItem("Show
Scale Overlay");
        showScaleItem.addActionListener(new GenericAction(this, "showScale"));
        JRadioButtonMenuItem hideScaleItem = new JRadioButtonMenuItem("Hide
Scale Overlay");
        hideScaleItem.addActionListener(new GenericAction(this, "hideScale"));
        JRadioButtonMenuItem showUnitItem = new JRadioButtonMenuItem("Show Unit
Overlay");
        showUnitItem.addActionListener(new GenericAction(this, "showUnit"));
        JRadioButtonMenuItem hideUnitItem = new JRadioButtonMenuItem("Hide Unit
Overlay");
        hideUnitItem.addActionListener(new GenericAction(this, "hideUnit"));
        JRadioButtonMenuItem showMapItem = new JRadioButtonMenuItem("Show Map");
        showMapItem.addActionListener(new GenericAction(this, "showMap"));
        JRadioButtonMenuItem hideMapItem = new JRadioButtonMenuItem("Hide Map");
        hideMapItem.addActionListener(new GenericAction(this, "hideMap"));
        overlayMenu.add(showCoastItem);
        showCoastItem.setSelected(true);
        overlayMenu.add(hideCoastItem);
        overlayMenu.addSeparator();
        coastbg.add(showCoastItem);
```

```
        coastbg.add(hideCoastItem);
        overlayMenu.add(showScaleItem);
        showScaleItem.setSelected(true);
        overlayMenu.add(hideScaleItem);
        overlayMenu.addSeparator();
        scalebg.add(showScaleItem);
        scalebg.add(hideScaleItem);
        overlayMenu.add(showUnitItem);
        showUnitItem.setSelected(true);
        overlayMenu.add(hideUnitItem);
        overlayMenu.addSeparator();
        unitbg.add(showUnitItem);
        unitbg.add(hideUnitItem);
        overlayMenu.add(showMapItem);
        showMapItem.setSelected(true);
        overlayMenu.add(hideMapItem);
        mapbg.add(showMapItem);
        mapbg.add(hideMapItem);

        JMenu mapMenu = new JMenu("Map");
        menu.add(mapMenu);
        mapAction = new GenericAction(this, "addNewMap");
        mapMenu.add(mapAction);

        JMenu helpMenu = new JMenu("Help");
        menu.add(helpMenu);
        JMenuItem contentsItem = new JMenuItem("Contents");
        JMenuItem searchItem = new JMenuItem("Search");
        helpMenu.add(contentsItem);
        helpMenu.add(searchItem);
        helpMenu.addSeparator();
        JMenuItem aboutItem = new JMenuItem("About ScenarioBuilder...");
        helpMenu.add(aboutItem);

        this.setJMenuBar((JMenuBar) menu);

//set the background map, and the overlays
        map = new ImageIcon(getClass().getResource("Ca.gif"));
        l = new JLabel();
        setMap(map);
        pan.add(l);
        l.add(scalePanel);
        l.add(p2);
        l.add(pan2);
        p.setViewportView(pan);
    }

    public ScenarioBuilderGUI() {
        this("Scenario Builder", "units.INI");
    }

//instance methods

/**
  * addNewMap replaces the current map with a gif or
  * jpg file.
**/
    public void addNewMap() {
        JFileChooser chooser = new JFileChooser();
```

```
    public void setModelDimensions() {
        f.setVisible(true);
    }

    public void storeModelDim() {
        getDimWC().cancel();
    }
**/

/**
  * setLatLong allows the user to set the Latitudinal
  * and Longitudinal scale of the model by entering two
  * points of Lat and Long.
**/
    public void setLatLong() {
        scalePanel.addMouseListener((MouseListener) scalePanel);
        scalePanel.addMouseMotionListener((MouseMotionListener) scalePanel);
    }

    public Action getNewCoast() {return newCoast;}

    public Action getEndCoast() {return endCoast;}

    public Action getEndIsle() {return endIsle;}

    public ImageIcon getMap() {return map;}

    public void showCoast() {
        p2.setVisible(true);
    }

    public void hideCoast() {
        p2.setVisible(false);
    }

    public void showScale() {
        scalePanel.setVisible(true);
    }

    public void hideScale() {
        scalePanel.setVisible(false);
    }

    public void showUnit() {
        pan2.setVisible(true);
    }

    public void hideUnit() {
        pan2.setVisible(false);
    }

    public void showMap() {
        l.setEnabled(true);
    }

    public void hideMap() {
        l.setEnabled(false);
    }
```

```
    public JComponent getP2() {return p2;}

    public JTextField getYField() {return yField;}

    public JTextField getXField() {return xField;}

    public WindowCloser getDimWC() {return dimWC;}

//main method

    public static void main(String[] args) {
       JFrame f = new ScenarioBuilderGUI();
       f.setSize(500, 500);
       f.setVisible(true);
    }
}
```

LIST OF REFERENCES

Clancy, Tom, *Marine: A Guided Tour of a Marine Expeditionary Unit,* Berkley Books, New York, NY, 1996.

Department of the Navy, *...From the Sea, Preparing the Naval Service for the 21st Century*, U.S. Government Printing Office, Washington D.C., 1992.

Department of the Navy, *Forward...From the Sea, Continuing the Preparation of the Naval Services for the 21st Century*, U.S. Government Printing Office, Washington D.C., 1994.

Dunnigan, James F., *How to Make War: A Comprehensive Guide to Modern Warfare*, William Morrow, New York, NY, 1988.

Geary, David M., *Graphic Java, Mastering the AWT*, Prentice Hall PTR, Upper Saddle River, NJ, 1997.

Hoeber, Francis P., *Military Applications of Modeling: Selected Case Studies*, Gordon and Breach Science Publishers, New York, NY, 1982.

Horstmann, Cay S. and Cornell, Gary, *Core Java 1.1, Volume I, Fundamentals*, Sun Microsystems Press, Mountain View, CA, 1997.

Horstmann, Cay S. and Cornell, Gary, *Core Java 1.1, Volume II, Advanced Features*, Sun Microsystems Press, Mountain View, CA, 1998.

Jaiswal, N. K., *Military Operations Research: Quantitative Decision Making*, Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

Mitchell, Mark L., *PRO-LOG Player's Manual*, Operations Research Dept. Naval Postgraduate School, Monterey, CA, 1988.

Naval Expeditionary Logistics Committee, *Naval Expeditionary Logistics: Enabling Operational Maneuver From the Sea*, National Academy Press, Washington D.C., 1999.

Schrady, D. A., Smyth G. K., & Vassin, R. B., *Predicted Ship Fuel Consumption: Update,* Operations Research Dept. Naval Postgraduate School, Monterey, CA, 1996.

Stork, Kirk A., *Sensors in Object Oriented Discrete Event Simulation*, Operations Research Dept. Naval Postgraduate School, Monterey, CA, 1996.

Topley, Kim, *CORE Java Foundation Classes*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.

Troxell, Anthony, *Naval Logistics Simulator*, Masters Thesis, Operations Research Department, Naval Postgraduate School, Monterey, CA, 1999.

Joint Staff, *Joint Vision 2010*, U.S. Government Printing Office, Washington D.C., 1996

Headquarters, U.S. Marine Corps, *Operational Maneuver from the Sea*, U. S. Government Printing Office, Washington D. C., 1996

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8275 John J Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411Dyer Rd.
   Monterey, California 93943-5101

3. Defense Logistic Studies Information Exchange 1
   U.S. Army Logistics Management Center
   Fort Lee, VA 23801-6043

4. Deputy Chief of Naval Operations (Logistics) 1
   Attn: CDR Carolyn Kresek, N421C
   2000 Navy Pentagon
   Washington, DC 20350-2000

5. Professor David Schrady, Code OR/SO 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Professor Arnold Buss, Code OR/BU 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. CDR Kevin J. Maher, Code OR/MK 1
   Department of Operations Research
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. LT John R. Sterba 3
   73 Cogswell Ct.
   Westerville, OH 43081